Overview and New Developments in Global Arrays

New Features of the Global Arrays Toolkit

Bruce Palmer

Jarek Nieplocha, Robert Harrison, Manoj Kumar Krishnan, Vinod Tipparaju, Harold Trease

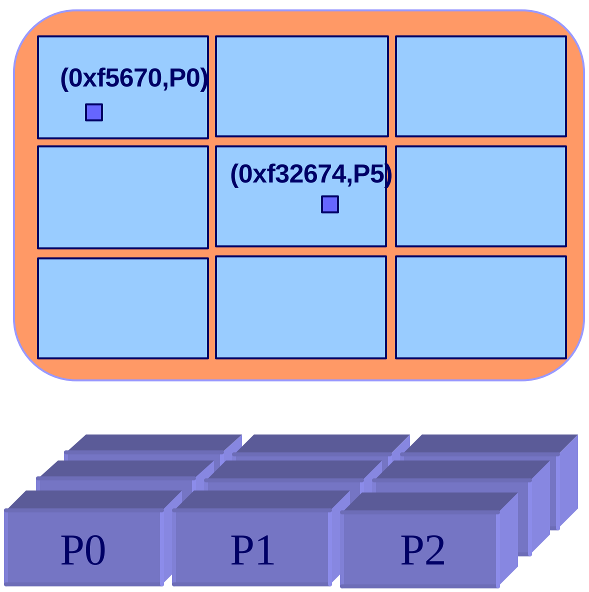
Pacific Northwest National Laboratory

Overview

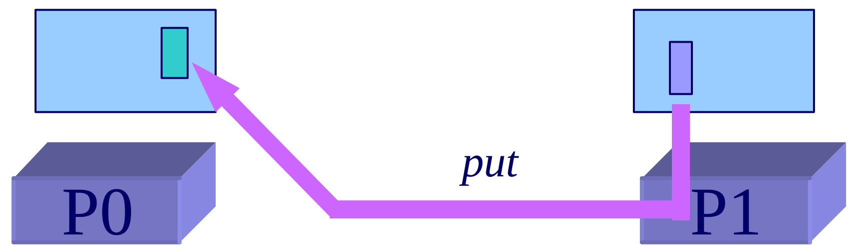
- Background
- Programming Model
- Core Capabilities
- Recent Work
- Future Directions

Global address space & One-sided communication

collection of address spaces of processes in a parallel job (address, pid)

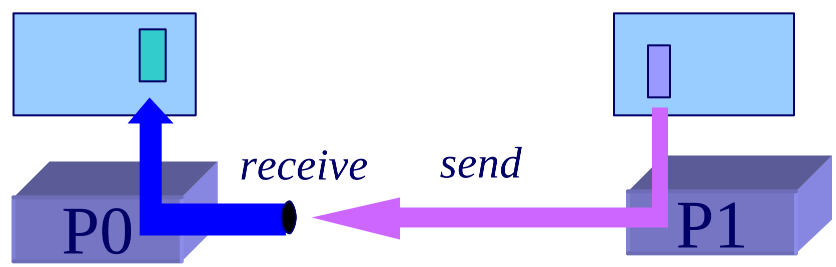


Communication models



one-sided communication
SHMEM, ARMCI, MPI-2-1S

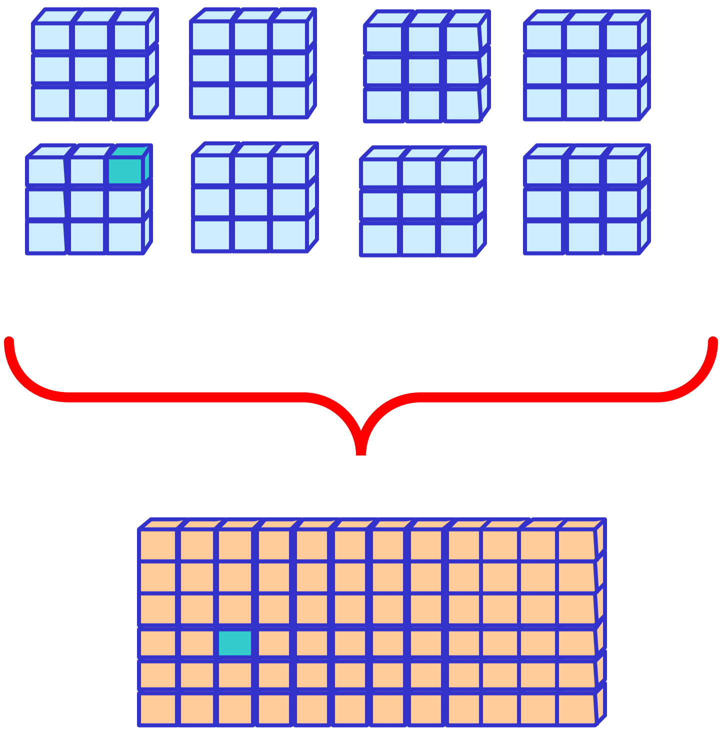
But not



message passing

Global Arrays Data Model

Physically distributed data

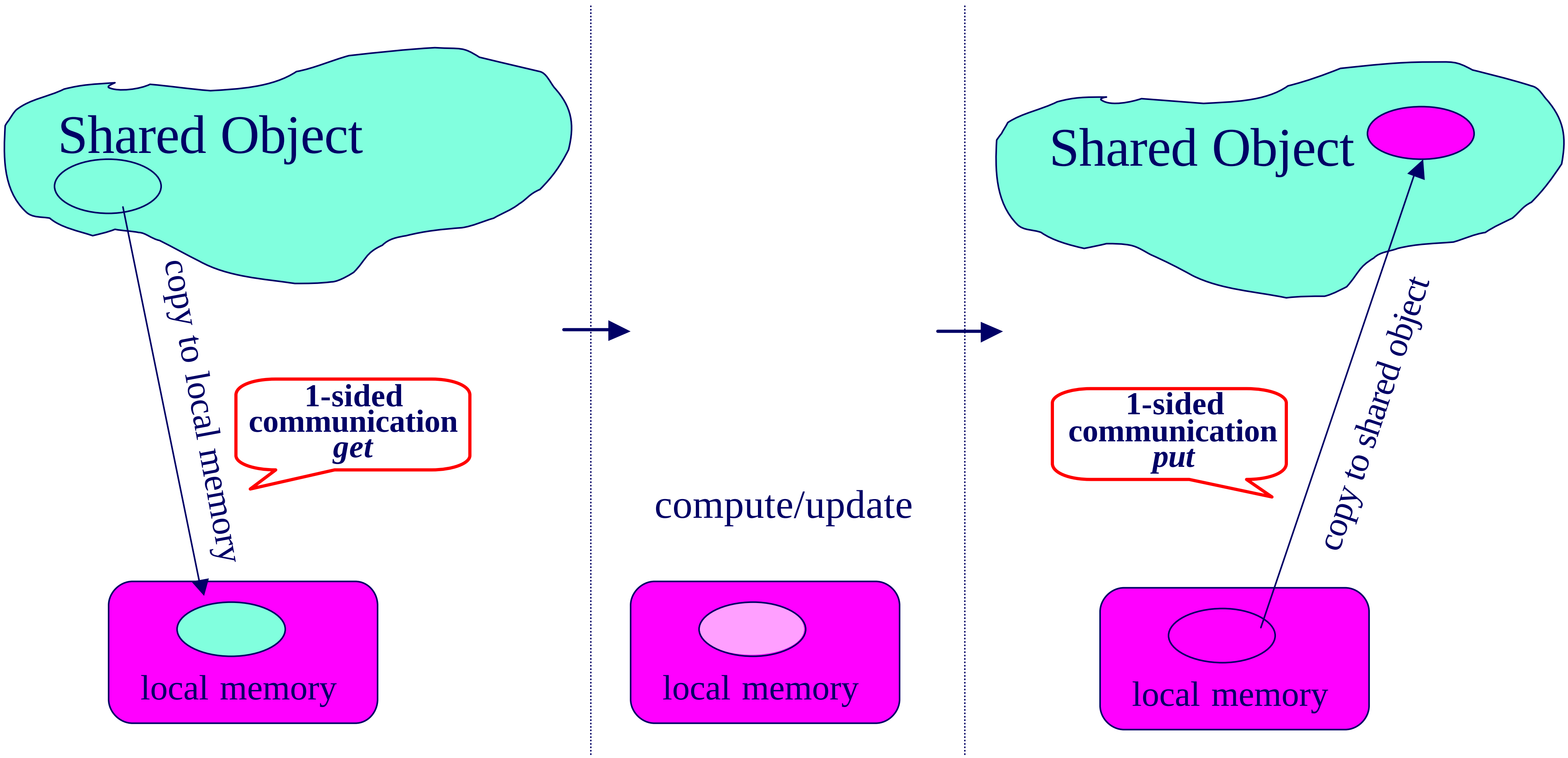


- shared memory model in context of distributed dense arrays
- complete environment for parallel code development
- compatible with MPI
- data locality control similar to distributed memory/message passing model
- extensible

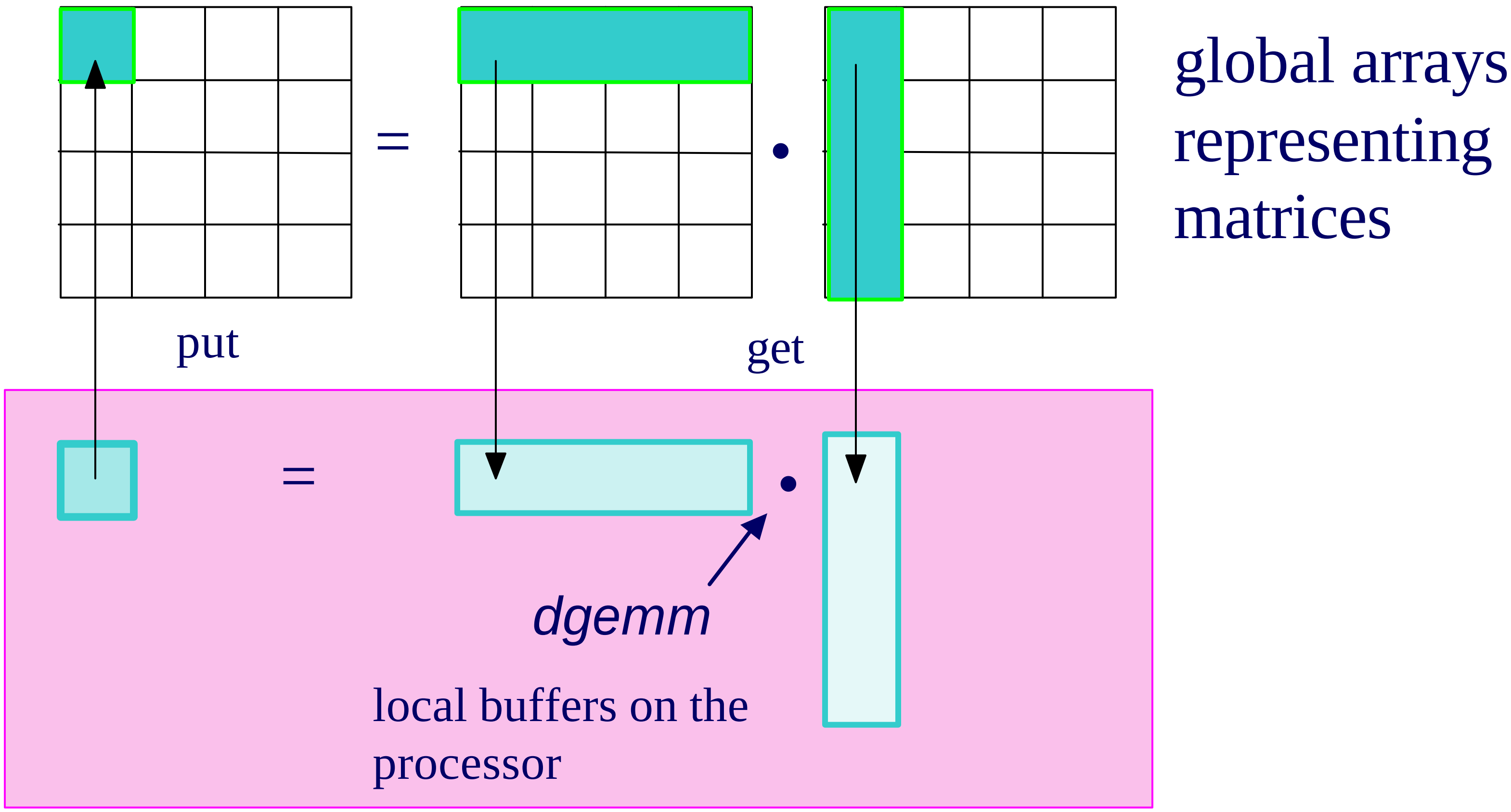
single, shared data structure/ global indexing

e.g., A(4,3) rather than buf(7) on task 2

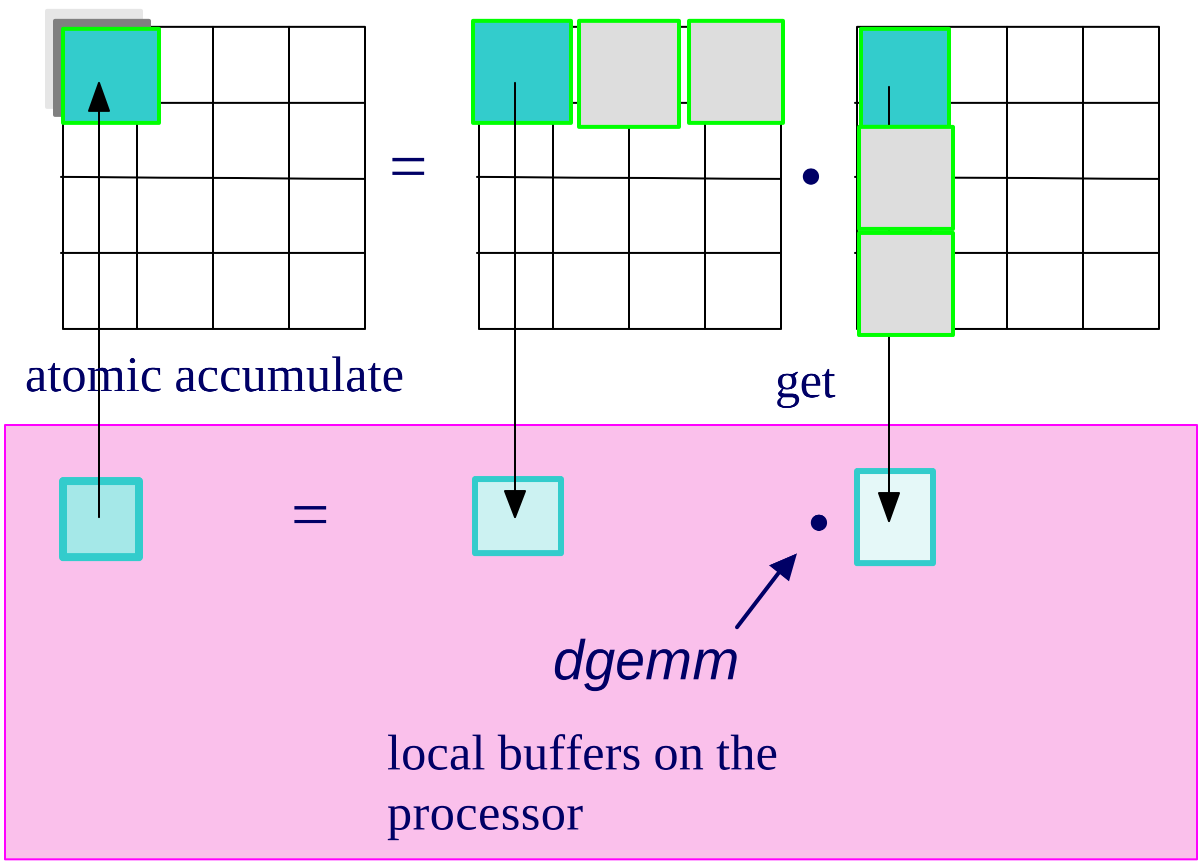
Global Array Model of Computations



Example: Matrix Multiply



Matrix Multiply (a better version)



more scalable!
(less memory, higher parallelism)

Comparison to other models

	Shared memory	Message passing	Global Arrays
Data view	shared	distributed	distributed **or** shared
Access to data	simplest (*a*=*b*)	hard (*send-receive*)	simple (*ga_put/get*)
Data locality information	obscure	explicit	easily available (*ga_disitribution/ ga_locate*)
Scalable performance	limited	very good	very good

Structure of GA

application interfaces
Fortran 77, C, C++, Python

distributed arrays layer
memory management, index translation

Message Passing
process creation,
run-time environment

ARMCI
portable 1-sided communication
put,get, locks, etc

system specific interfaces
LAPI, GM/Myrinet, threads, VIA,..

Core Capabilities

- Distributed array library
  - dense arrays 1-7 dimensions
  - four data types: *integer*, *real, double precision*, *double complex*
  - global rather than per-task view of data structures
  - user control over data distribution: regular and irregular
- Collective and shared-memory style operations
- Interfaces to third party parallel numerical libraries
  - PeIGS, Scalapack, SUMMA, Tao
    - example: to solve a linear system using LU factorization

```
call ga_lu_solve(g_a, g_b)
```

instead of

```
call pdgetrf(n,m, locA, p, q, dA, ind, info)
call pdgetrs(trans, n, mb, locA, p, q, dA,dB,info)
```

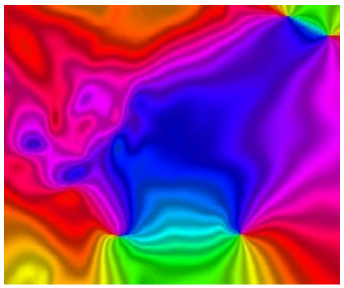
Performance

- Performance model for remote data access
  - array index translation e.g., 1.2 ?S on Linux/PIII
  - overhead in one of more ARMCI put/get/… calls
    - direct mapping to native RMA calls (e.g., 3?S on Cray T3E) or
    - simple shared memory access (e.g., 0.3 ?S on Linux/PIII) or
    - more complex due to the Active Message style implementations e.g. 12 (put) 37 (get) ?S on Linux/PIII with Myrinet

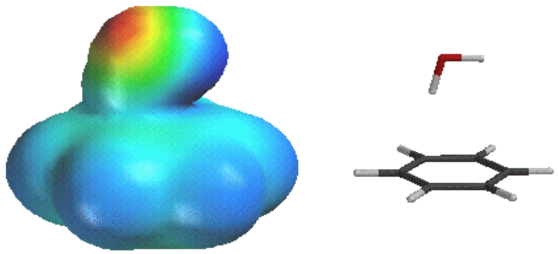
Interoperability and Interfaces

- GA provides a set of operations exposing
  - data in global arrays on individual processes, memory layout
  - array distribution information and process mapping
- Interoperability with MPI libararies
  - e.g., PETSC, CUMULVS
- Explicit interfaces to other systems that expand functionality of GA
  - ScaLAPACK-scalable linear algebra software
  - Peigs-parallel eigensolvers
  - TAO-advanced optimization package

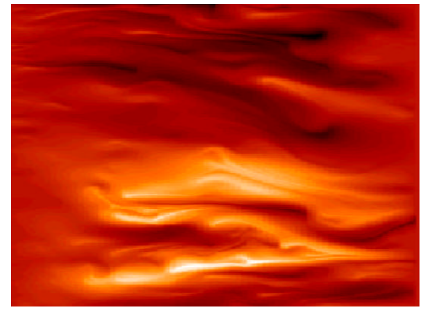
Applications Areas



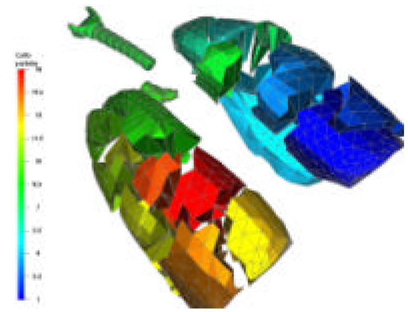
Visualization and image analysis



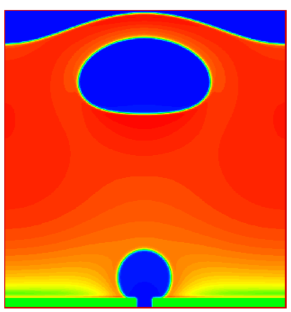
electronic structure



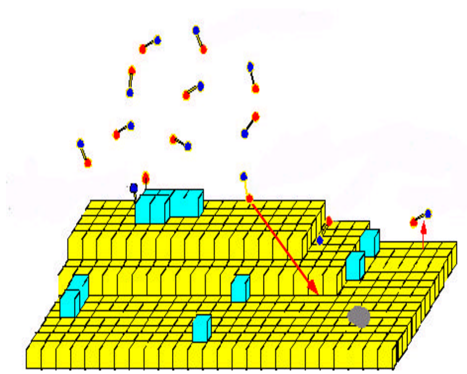
glass flow simulation



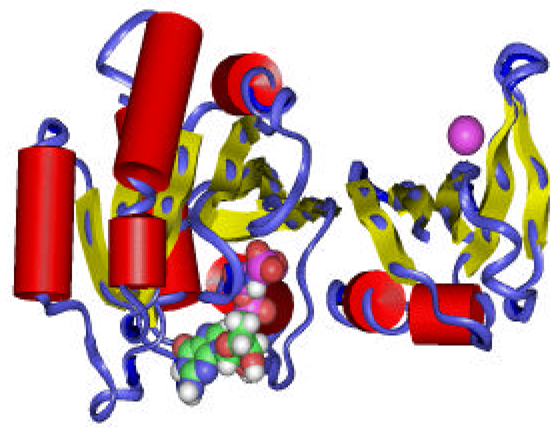
biology



thermal flow simulation



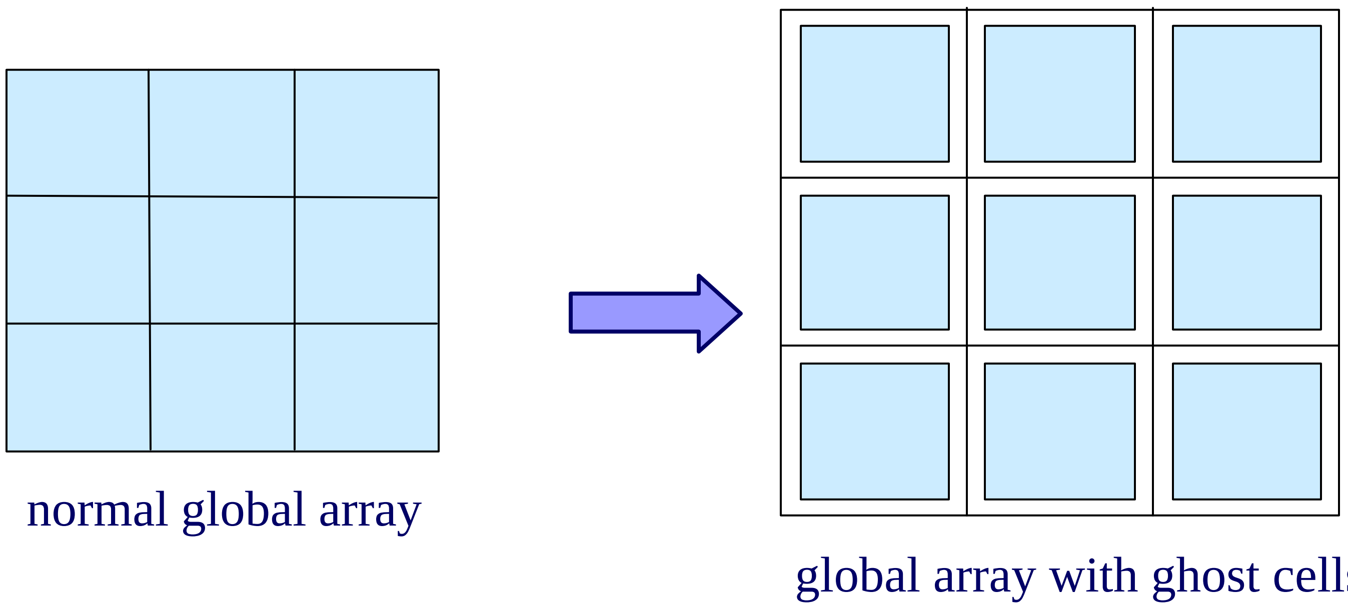
material sciences



molecular dynamics

Others: financial security forecasting, astrophysics, geosciences

Ghost Cells



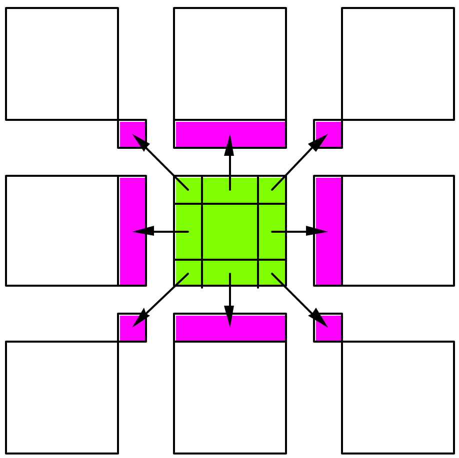
- **Operations**

  `NGA_Create_ghosts` - creates array with ghosts cells

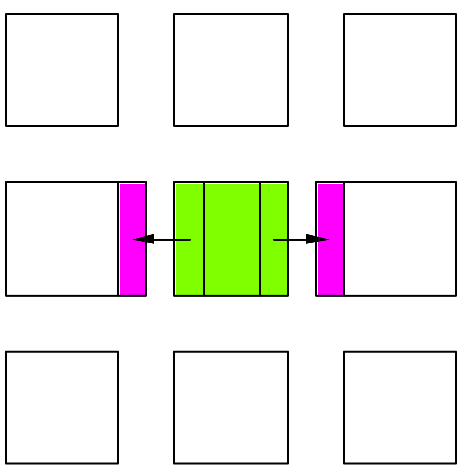
  `GA_Update_ghosts` - updates with data from adjacent processors

  `NGA_Access_ghosts` - provides access to “local” ghost cell elements
- **Embedded Synchronization - controlled by the user**
- **Multi-protocol implementation to match platform characteristics**
  - **e.g., MPI+shared memory on the IBM SP, SHMEM on the Cray T3E**

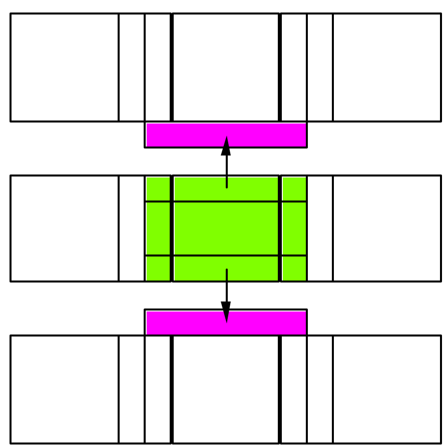
Update Algorithms



- **Standard algorithm: 3^D-1 messages**



1st phase

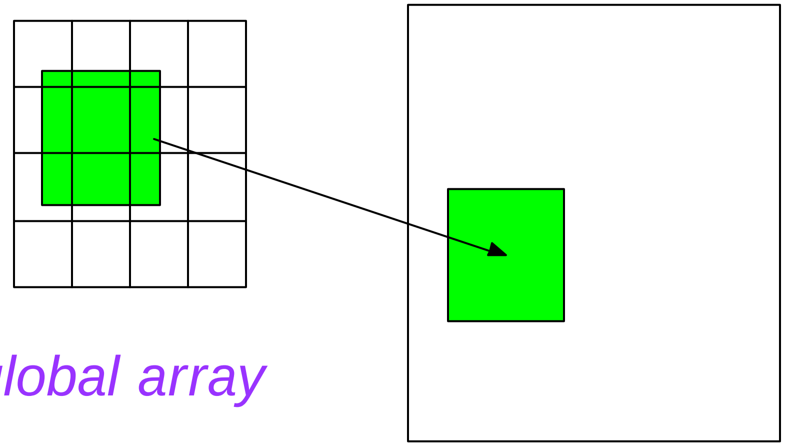


2nd phase

- **Shift algorithm: 2D messages**

Disk Resident Arrays

- Extend GA model to disk
  - system similar to Panda (U. Illinois) but higher level APIs
- Provide easy transfer of data between N-dim arrays stored on disk and stored in memory
- Use when
  - Arrays too big to store in core
  - checkpoint/restart
  - out-of-core solvers



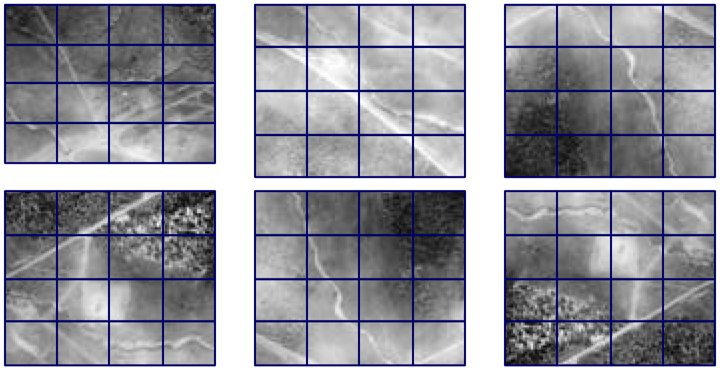
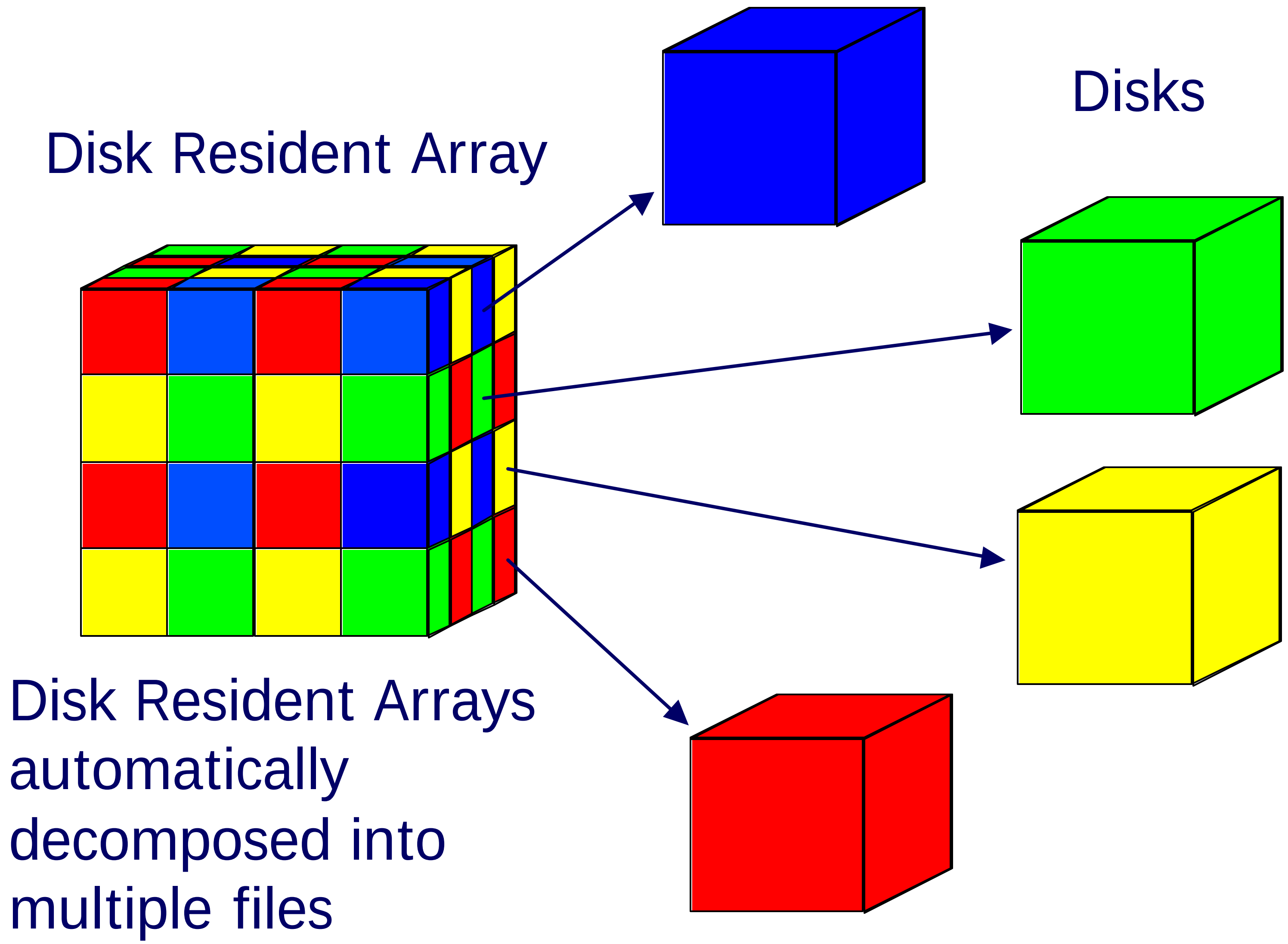
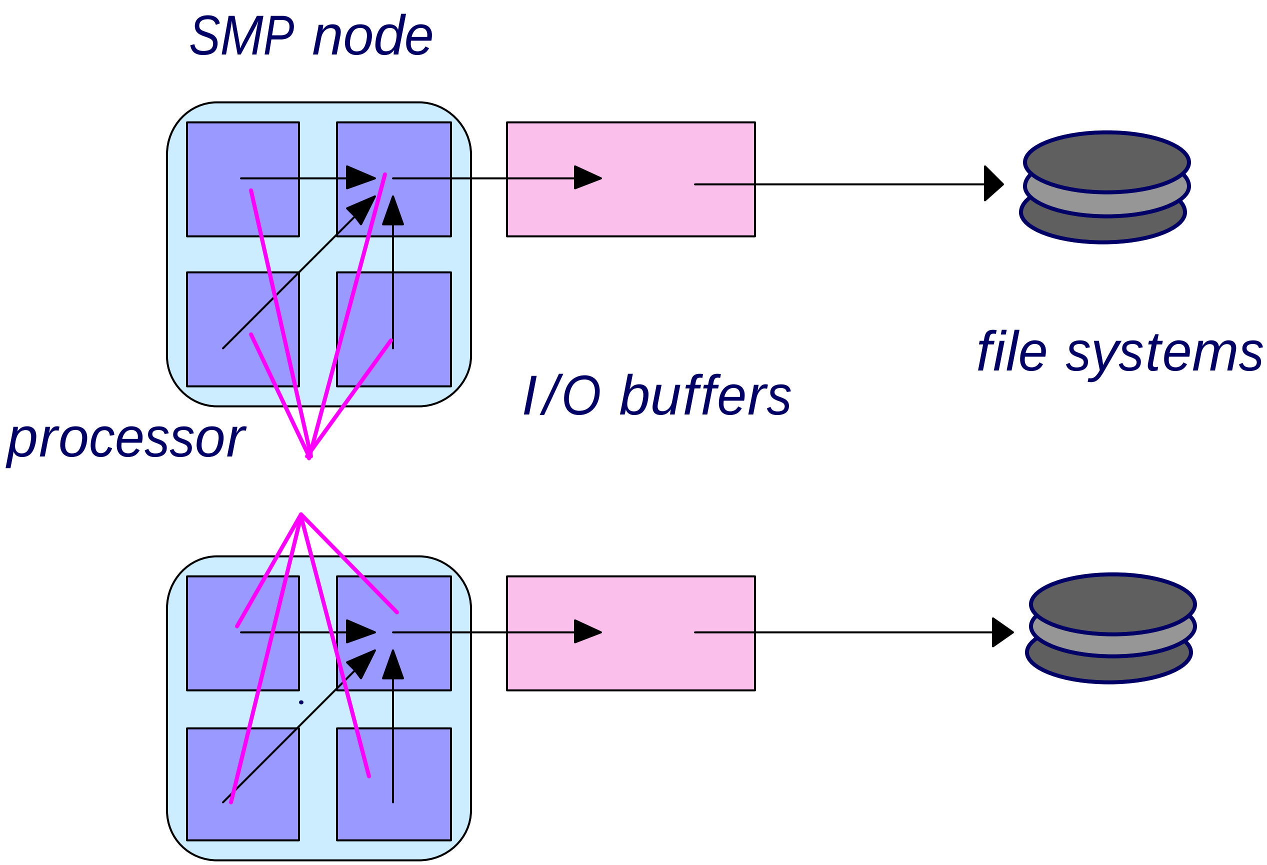


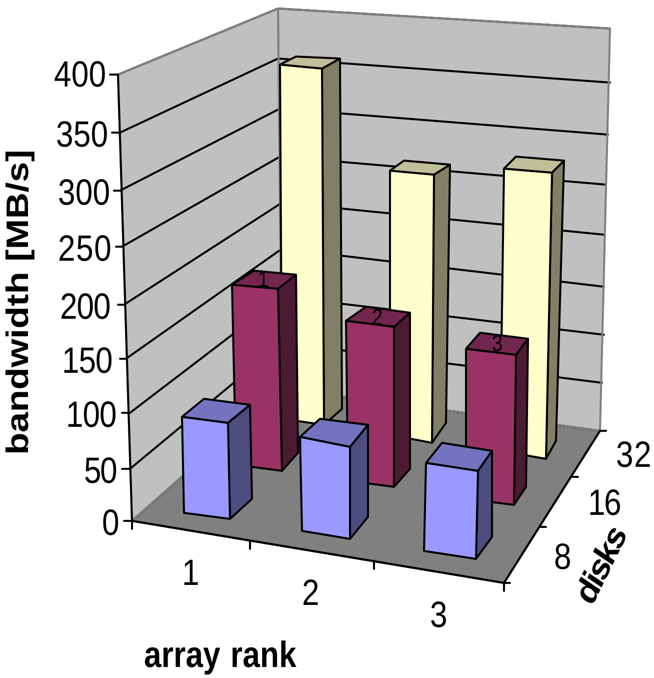
image processing application

High Bandwidth Read/Write



Scalable Performance of DRA

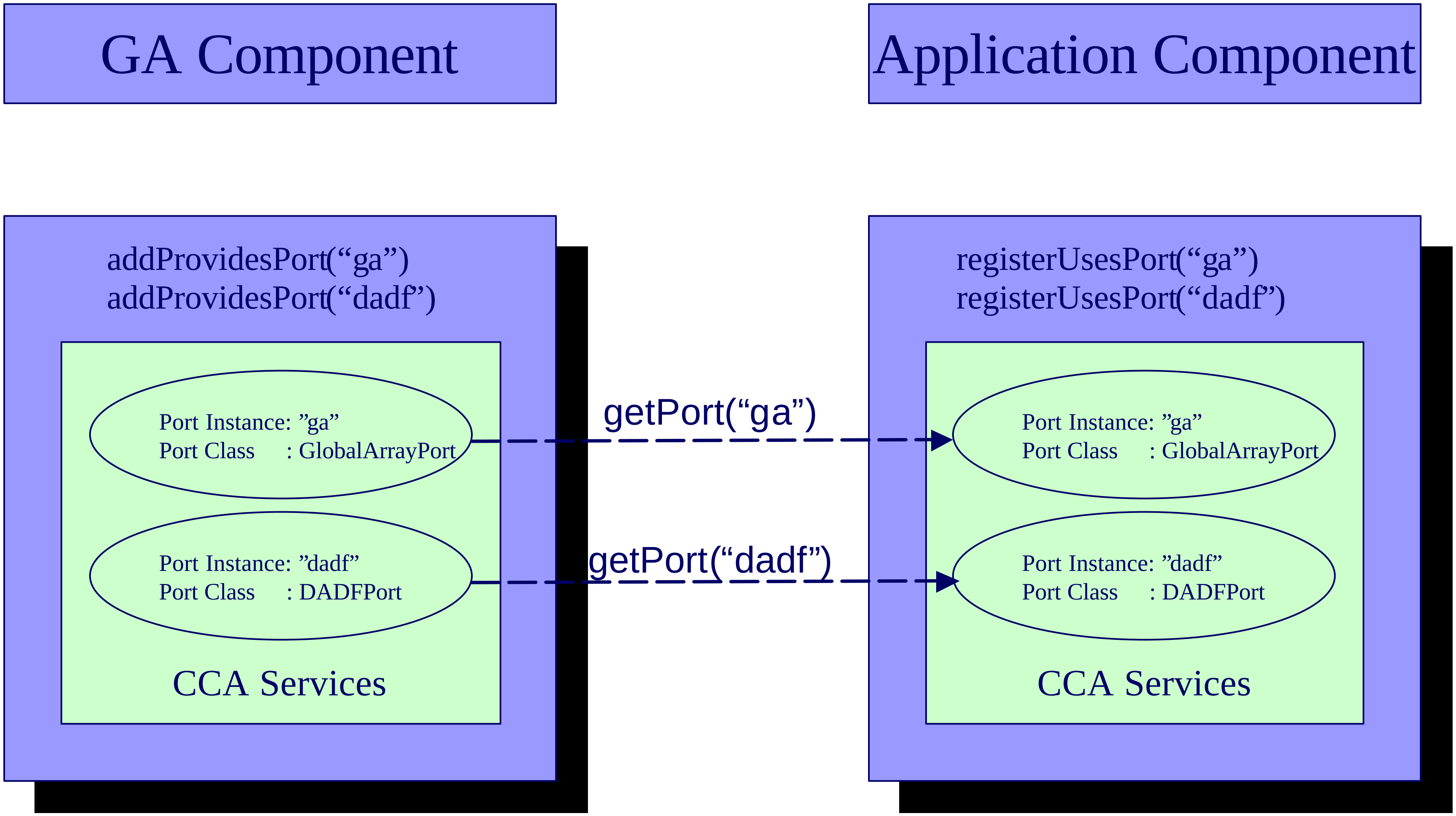




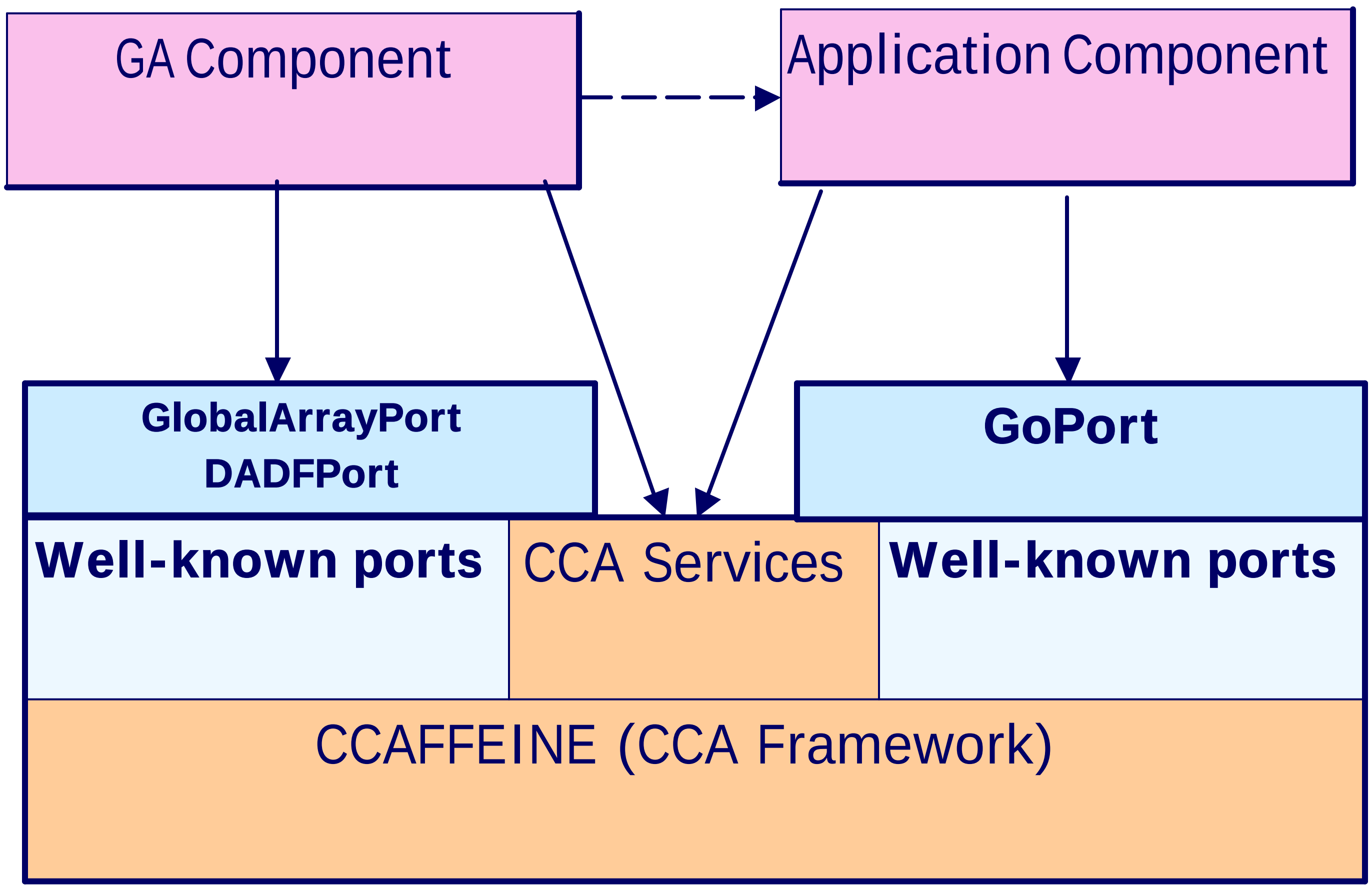
Common Component Architecture

- A component model specifically designed for HPC
  - Three parts: Components, Ports and Frameworks
- Components
  - peers
  - interact through well-defined interfaces (ports)
    - In OO Language a port is a class
    - In Fortran, a port is a bunch of subroutines
  - A component may provide a port - implement the class/subroutines
  - Another component may use that port – call methods in the port
- Framework holds the components and compose them into "applications"
- Advantages: Reusable functionality, well-defined interfaces, etc.

Global Array CCA Component



CCA Elements



GA-CCA Component

- Two flavors of GA Component
  - CCAFFEINE Framework (SNL)
  - Decaf Framework (LLNL)
- Common Component Architecture (CCA) compliant
- Decaf uses SIDL interfaces for components
  - Language Interoperability
- Current Work
  - Build GA Component using SIDL interface in CCAFFEINE framework
  - Integrate with TAO component (ANL)

GA++

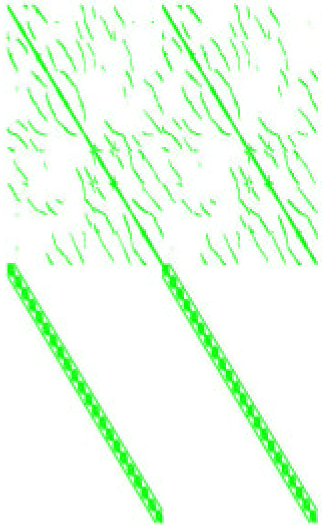
- GA++ is a C++ class library for Global Arrays
- GA++ classes: GAservices, GlobalArray

GAservices → Initialization, Termination, Inter-process Synchronization, etc

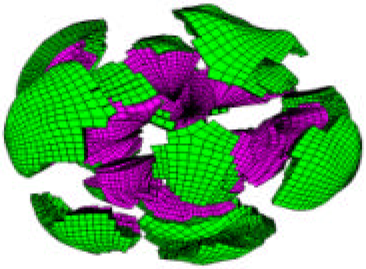
GlobalArray → One-sided(get/put), collective array, Utility operations

```
GAservices gs;
gs.initialize();
Global Array *ga=gs->createGA(…)
    …do work …
ga->destroy();
gs.terminate();
```

Sparse data managment



- Sparse arrays can be implemented with
  - 1-dimensional global arrays
    - Nonzero elements, row and/or index arrays
  - Set of new operations that follow Thinking Machines CMSSL
    - Enumerate
    - Pack/unpack
    - Binning (NxM mapping)
    - 2-key binning/sorting functions
    - Scatter_with_OP, where OP={+,min,max}
    - Segmented_scan_with_OP, where OP={+,min,max,copy}



- Adopted in NWPhys/NWGrid AMR package
- Next step - explicit sparse format
  - need more application experience - too many degrees of freedom

Summary and Future

- The idea proven very successful
  - efficient on a wide range of architectures
    - core operations tuned for high performance
  - library substantially extended but all original (1994) APIs preserved
  - increasing number of application areas
- Ongoing and future work
  - Latency hiding on the low-end cluster networks by relaxed memory consistency and replication
  - Advanced data structures
    - sparse arrays and hash tables
  - Increased support for the HPC community standards
    - ESI, CCA

Major Milestones

- 1994 - 1st public release of GA
- 1995 - Metacomputing (grid) extensions of GA
- 1996 - DRA, parallel I/O for GA programs developed
- 1997 - development of ARMCI started
- 1998 - GA rewritten to use ARMCI
- 1999 - GA 3.0 released, n-dimensional arrays
- 2000 - periodic one-sided operations
- 2001 - support for sparse data management
- 2002 - ghost cell operations, n-dim DRA

Source Code and More Information

- Version 3.2 available in beta release
- Homepage at http://www.emsl.pnl.gov:2080/docs/global/
- Platforms
  - IBM SP
  - Cray T3E, SV1
  - Linux Cluster with Myrinet or Ethernet
  - SGI/Irix
  - Solaris
  - Fujitsu
  - Hitachi
  - NEC
  - Compaq
  - Windows